

Computing the eigenvalues of symmetric \mathcal{H}^2 -matrices by slicing the spectrum ^{*}

Peter Benner Steffen Börm Thomas Mach [†] Knut Reimer [‡]

July 16, 2014

The computation of eigenvalues of large-scale matrices arising from finite element discretizations has gained significant interest in the last decade [21]. Here we present an new algorithm based on slicing the spectrum that takes advantage of the rank structure of resolvent matrices in order to compute m eigenvalues of the generalized symmetric eigenvalue problem in $\mathcal{O}(nm \log^\alpha n)$ operations, where $\alpha > 0$ is a small constant.

1 Introduction

The numerical solution of the generalized eigenproblem

$$(A - \lambda B)x = 0, \tag{1}$$

given $A, B \in \mathbb{R}^{n \times n}$ and searching for $\lambda \in \mathbb{C}$ and $x \in \mathbb{C}^n \setminus \{0\}$, is one of the fundamental problems in the computational sciences and engineering. It arises in numerous applications ranging from structural and vibrational analysis to problems in computational physics and chemistry like electronic and band structure calculations, see, e.g., [21] and the reports therein. In particular, the investigation and design of new materials poses numerous new challenges for the numerical solution of (1). These include the necessity to compute more than just the (few) smallest magnitude eigenvalue(s) — the target of many algorithms discussed in the literature. Often in these problems, a large number of interior eigenvalues are required. This poses a significant challenge for most popular algorithms used to solve large-scale eigenproblems based on the Arnoldi or

^{*}

[†]The research of the third author was partially supported by the Research Council KU Leuven, fellowship F4/13/020 Exploiting Unconventional QR-Algorithms for Fast and Accurate Computations of Roots of Polynomials; and by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office, Belgian Network DYSCO (Dynamical Systems, Control, and Optimization).

[‡]The research of the second and fourth authors was funded by the DFG Deutsche Forschungsgemeinschaft in project BO 3289/4-1.

Lanczos processes or the Jacobi-Davidson method. Therefore, we will discuss here a different approach that has received little attention in the literature so far: the slicing-the-spectrum approach discussed in [24].

Many of the application problems listed above lead to a symmetric eigenproblem in the sense that $A = A^T$ and $B = B^T$. Moreover, in applications arising from the discretization of (elliptic) partial differential operators — which probably cover the majority of these application problems — the matrix B is a mass matrix and thus positive definite, which we denote by $B > 0$. In this situation, the eigenvalues λ and eigenvectors x are all real. Here, we will assume these conditions and furthermore, we will focus on the computation of eigenvalues. If necessary, eigenvectors corresponding to selected eigenvalues can be computed by inverse iteration which we will not further discuss.

Slicing-the-spectrum allows to compute a selected number of eigenvalues of a symmetric matrix, or even all of them. It requires knowledge of the inertia of shifted versions of the matrix, which can be computed by the LDL^T factorization. As this is quite an expensive computation for sparse matrices, the method has received little attention in the literature. For data-sparse matrices which allow a low complexity computation of the LDL^T factorization, though, this method becomes attractive again. In [4], we have used this approach to show that some, say m , eigenvalues of \mathcal{H}_ℓ -matrices can be computed in $\mathcal{O}(mn \log^\alpha(n))$ complexity (for a discussion of the involved constants we refer to [4]). \mathcal{H}_ℓ -matrices are a class of simple hierarchical (\mathcal{H} -) matrices that are rank-structured in their off-diagonal parts. That is, the off-diagonal parts of these matrices are represented in a hierarchical way by low-rank blocks so that the total storage for the matrix is of linear-logarithmic complexity. Such matrices often arise from the discretization of non-local operators arising in integral equations or as solution operators of (elliptic) partial differential operators [12, 14, 16, 7, 15], and can therefore often be used in the above application problems for the algebraic representation of the involved integral and differential operators.

It was shown in [4] that the LDL^T factorization for \mathcal{H}_ℓ -matrices has bounded block ranks. This allows the efficient implementation of the slicing-the-spectrum approach for these special \mathcal{H} -matrices. Numerical experiments however illustrated that this does not hold for \mathcal{H} -matrices, casting doubt on the usefulness of this approach for more general rank-structured matrices. In this paper, we investigate the slicing-the-spectrum approach for \mathcal{H}^2 -matrices. This matrix format allows a further compression compared to \mathcal{H} -matrices by considering the low-rank structure of the whole off-diagonal part of a block-row rather than of individual blocks. We will see that an efficient LDL^T factorization of \mathcal{H}^2 -matrices is possible and thus, an efficient implementation of the slicing-the-spectrum approach is feasible. We will furthermore extend this approach from the standard eigenvalue problem considered in [4] to the symmetric-definite eigenproblem (1) with A, B symmetric and $B > 0$. Moreover, this approach is shown to be easily parallelizable which allows to gain further efficiency on current computer architectures.

The paper is structured as follows: in Section 2, we introduce the necessary background on \mathcal{H}^2 -matrices. We then discuss the efficient implementation of the LDL^T factorization in the \mathcal{H}^2 -format. The slicing-the-spectrum approach is then reviewed in Section 3. Furthermore, the application to \mathcal{H}^2 -matrices is discussed as well as the extension to the symmetric-definite eigenproblem. We also discuss a parallel implementation of the method. Numerical experiments illustrating the performance of the \mathcal{H}^2 -slicing-the-spectrum algorithm and its parallelization are

presented in Section 4.

2 \mathcal{H}^2 -Matrices and Their LDL^T Factorization

2.1 \mathcal{H}^2 -matrices

In this section we briefly recollect the basic definitions of \mathcal{H}^2 -matrices [19, 8]: matrices are split into submatrices according to a *block tree*, and this tree is constructed using *cluster trees* describing the decomposition of row and column index sets. If a submatrix is *admissible*, it is represented in factorized form using low-rank *cluster bases* and *coupling matrices*.

Definition 1 (Cluster tree) Let \mathcal{I} be a finite index set, and let \mathcal{T} be a labeled tree. Denote the label of each node $t \in \mathcal{T}$ by \hat{t} .

\mathcal{T} is called a *cluster tree* for \mathcal{I} if the following conditions hold:

- its root $r = \text{root}(\mathcal{T})$ is labeled by \mathcal{I} , i.e., $\hat{r} = \mathcal{I}$,
- for all $t \in \mathcal{T}$ with $\text{sons}(t) \neq \emptyset$, we have $\hat{t} = \bigcup_{t' \in \text{sons}(t)} \hat{t}'$,
- for all $t \in \mathcal{T}$ and all $t_1, t_2 \in \text{sons}(t)$, $t_1 \neq t_2$, we have $\hat{t}_1 \cap \hat{t}_2 = \emptyset$.

A cluster tree for \mathcal{I} is denoted by $\mathcal{T}_{\mathcal{I}}$, its nodes are called *clusters*, and $\mathcal{L}_{\mathcal{I}} := \{t \in \mathcal{T}_{\mathcal{I}} : \text{sons}(t) = \emptyset\}$ defines the set of its leaves.

Remark 2 (Leaf partition) The definition implies $\hat{t} \subseteq \mathcal{I}$ for all $t \in \mathcal{T}_{\mathcal{I}}$.

We also have that the labels of the leaves of $\mathcal{T}_{\mathcal{I}}$ form a disjoint partition $\{\hat{t} : t \in \mathcal{L}_{\mathcal{I}}\}$ of the index set \mathcal{I} [18, 8].

Remark 3 (Cardinalities) Let $n_{\mathcal{I}} := \#\mathcal{I}$ denote the number of indices. In typical situations, a cluster tree consists of $\mathcal{O}(n_{\mathcal{I}}/k)$ clusters, where k denotes the rank used to approximate matrix blocks.

The sum of the cardinalities of the index sets corresponding to all clusters is typically in $\mathcal{O}(n_{\mathcal{I}} \log(n_{\mathcal{I}}))$ [8], since each index appears in $\mathcal{O}(\log(n_{\mathcal{I}}))$ clusters.

Remarks 2 and 3 imply that algorithms with optimal (linear) complexity should have at most constant complexity in all non-leaf clusters $t \in \mathcal{T}_{\mathcal{I}} \setminus \mathcal{L}_{\mathcal{I}}$ and linear complexity (with respect to $\#\hat{t}$) in all leaf clusters $t \in \mathcal{L}_{\mathcal{I}}$.

With the help of the cluster tree we are able to define the block tree, which gives us a hierarchically structured block partition of $\mathcal{I} \times \mathcal{J}$ and ultimately a partition of matrices $X \in \mathbb{R}^{\mathcal{I} \times \mathcal{J}}$ into submatrices.

Definition 4 (Block tree) Let $\mathcal{T}_{\mathcal{I}}$ and $\mathcal{T}_{\mathcal{J}}$ be cluster trees for index sets \mathcal{I} and \mathcal{J} , respectively.

A labeled tree \mathcal{T} is called a *block tree* for $\mathcal{T}_{\mathcal{I}}$ and $\mathcal{T}_{\mathcal{J}}$ if the following conditions hold:

- for all $b \in \mathcal{T}$, there are $t \in \mathcal{T}_{\mathcal{I}}$ and $s \in \mathcal{T}_{\mathcal{J}}$ such that $b = (t, s)$ and $\hat{b} = \hat{t} \times \hat{s}$,

- the root is $r = \text{root}(\mathcal{T}) = (\text{root}(\mathcal{T}_{\mathcal{I}}), \text{root}(\mathcal{T}_{\mathcal{J}}))$,
- for all $b = (t, s) \in \mathcal{T}$, $\text{sons}(b) \neq \emptyset$, we have

$$\text{sons}(b) = \begin{cases} \{t\} \times \text{sons}(s) & \text{if } \text{sons}(t) = \emptyset \neq \text{sons}(s), \\ \text{sons}(t) \times \{s\} & \text{if } \text{sons}(t) \neq \emptyset = \text{sons}(s), \\ \text{sons}(t) \times \text{sons}(s) & \text{otherwise.} \end{cases}$$

A block tree for $\mathcal{T}_{\mathcal{I}}$ and $\mathcal{T}_{\mathcal{J}}$ is denoted by $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$, its nodes are called blocks, and the set of its leaves is denoted by $\mathcal{L}_{\mathcal{I} \times \mathcal{J}} := \{b \in \mathcal{T}_{\mathcal{I} \times \mathcal{J}} : \text{sons}(b) = \emptyset\}$.

For all blocks $b = (t, s) \in \mathcal{T}_{\mathcal{I} \times \mathcal{J}}$, t is called the row cluster and s is called the column cluster.

Remark 5 (Leaf partition) The Definitions 1 and 4 imply that a block tree $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ is a cluster tree for $\mathcal{I} \times \mathcal{J}$ and that therefore the set of the labels of its leaves $\{\hat{t} \times \hat{s} : b = (t, s) \in \mathcal{L}_{\mathcal{I} \times \mathcal{J}}\}$ is a disjoint partition of $\mathcal{I} \times \mathcal{J}$. We use this partition to split matrices into submatrices.

To determine which of these submatrices can be approximated by low-rank representations, we split the set $\mathcal{L}_{\mathcal{I} \times \mathcal{J}}$ of leaf blocks into a set of *admissible* blocks and a remainder of “sufficiently small” blocks.

Definition 6 (Admissible blocks) Let $\mathcal{L}_{\mathcal{I} \times \mathcal{J}}^+ \subseteq \mathcal{L}_{\mathcal{I} \times \mathcal{J}}$ be a subset of the leaves $\mathcal{L}_{\mathcal{I} \times \mathcal{J}}$ and let $\mathcal{L}_{\mathcal{I} \times \mathcal{J}}^- := \mathcal{L}_{\mathcal{I} \times \mathcal{J}} \setminus \mathcal{L}_{\mathcal{I} \times \mathcal{J}}^+$ denote the remaining leaves.

If $(t, s) \in \mathcal{L}_{\mathcal{I} \times \mathcal{J}}^+$ implies $t \in \mathcal{L}_{\mathcal{I}}$ and $s \in \mathcal{L}_{\mathcal{J}}$, we call $\mathcal{L}_{\mathcal{I} \times \mathcal{J}}^+$ a set of admissible blocks and $\mathcal{L}_{\mathcal{I} \times \mathcal{J}}^-$ the corresponding set of inadmissible blocks.

Typically we choose the set $\mathcal{L}_{\mathcal{I} \times \mathcal{J}}^+$ of admissible leaves in a way that ensures that for each $b = (t, s) \in \mathcal{L}_{\mathcal{I} \times \mathcal{J}}^+$, the corresponding submatrix $X_{\hat{t} \times \hat{s}}$ can be approximated by low rank. In practice a minimal block tree is constructed based on an *admissibility condition* that predicts whether a given block $b = (t, s)$ can be approximated. If this is the case, the block is chosen as an admissible leaf of $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$. Otherwise we either check the sons of b given by Definition 4 or, if there are no sons, declare the block an inadmissible leaf.

In the context of elliptic partial differential equations, we usually employ an *admissibility criterion* of the form

$$\max\{\text{diam}(t), \text{diam}(s)\} \leq 2\eta \text{dist}(t, s),$$

where $\text{diam}(t)$ and $\text{dist}(t, s)$ denote the diameter and distance of clusters in a suitable way.

Remark 7 (Sparse block tree) If there is a constant $c_{\text{sp}} \in \mathbb{N}$ such that

$$\begin{aligned} \#\{s \in \mathcal{T}_{\mathcal{J}} : (t, s) \in \mathcal{T}_{\mathcal{I} \times \mathcal{J}}\} &\leq c_{\text{sp}} & \text{for all } t \in \mathcal{T}_{\mathcal{I}}, \\ \#\{t \in \mathcal{T}_{\mathcal{I}} : (t, s) \in \mathcal{T}_{\mathcal{I} \times \mathcal{J}}\} &\leq c_{\text{sp}} & \text{for all } s \in \mathcal{T}_{\mathcal{J}} \end{aligned}$$

hold, we call the block tree $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ c_{sp} -sparse.

In this case, Remark 3 implies that the number of blocks $\#\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ is in $\mathcal{O}(n_{\mathcal{I}} + n_{\mathcal{J}})$, so algorithms of optimal complexity should require only a constant number of operations per block.

\mathcal{H}^2 -matrices use a three-term representation $V_t S_{(t,s)} W_s^T$ for all admissible blocks. The matrix V_t depends only on the row cluster t and W_s only on the column cluster s . The advantage of this representation is that only the $k \times k$ matrix $S_{(t,s)}$ is stored for every admissible block (t, s) .

Storing the matrices V_t and W_s directly would lead to linear complexity in each cluster. Thus we would get log-linear complexity for the whole families of matrices $(V_t)_{t \in \mathcal{T}_\mathcal{I}}$ and $(W_s)_{s \in \mathcal{T}_\mathcal{J}}$ (see Remark 3). In [17] the more efficient *nested* representation of these families is introduced.

Definition 8 (Cluster basis) Let $k \in \mathbb{N}$, and let $(V_t)_{t \in \mathcal{T}_\mathcal{I}}$ be a family of matrices satisfying $V_t \in \mathbb{R}^{\hat{t} \times k}$ for all $t \in \mathcal{T}_\mathcal{I}$.

This family is called a (nested) cluster basis if for each $t \in \mathcal{T}_\mathcal{I}$ there is a matrix $E_t \in \mathbb{R}^{k \times k}$ such that

$$V_{t|t' \times k} = V_{t'} E_{t'} \quad \text{for all } t \in \mathcal{T}_\mathcal{I}, t' \in \text{sons}(t). \quad (2)$$

The matrices E_t are called transfer matrices, and k is called the rank of the cluster basis.

Due to (2), we only have to store the $\hat{t} \times k$ matrices V_t for leaf clusters $t \in \mathcal{L}_\mathcal{I}$ and the $k \times k$ transfer matrices E_t for all clusters $t \in \mathcal{T}_\mathcal{I}$.

Remark 9 (Storage) According to Remark 2, the “leaf matrices” $(V_t)_{t \in \mathcal{L}_\mathcal{I}}$ require $n_\mathcal{I} k$ units of storage. The transfer matrices $(E_t)_{t \in \mathcal{T}_\mathcal{I}}$ require $k^2 \# \mathcal{T}_\mathcal{I}$ units of storage. With the standard assumption $\# \mathcal{T}_\mathcal{I} \lesssim n_\mathcal{I} / k$, we can conclude that a cluster basis can be represented in $\mathcal{O}(n_\mathcal{I} k)$ units of storage [17, 9, 8].

Definition 10 (\mathcal{H}^2 -matrix) Let $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ be cluster trees for index sets \mathcal{I} and \mathcal{J} , let $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ be a matching block tree, and let $(V_t)_{t \in \mathcal{T}_\mathcal{I}}$ and $(W_s)_{s \in \mathcal{T}_\mathcal{J}}$ be nested cluster bases.

A matrix $G \in \mathbb{R}^{\mathcal{I} \times \mathcal{J}}$ is called an \mathcal{H}^2 -matrix for $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$, $(V_t)_{t \in \mathcal{T}_\mathcal{I}}$ and $(W_s)_{s \in \mathcal{T}_\mathcal{J}}$ if for each admissible block $b = (t, s) \in \mathcal{L}_{\mathcal{I} \times \mathcal{J}}^+$ there is a matrix $S_b \in \mathbb{R}^{k \times k}$ such that

$$G_{|\hat{t} \times \hat{s}} = V_t S_b W_s^T. \quad (3)$$

The matrices S_b are called coupling matrices, the cluster bases $(V_t)_{t \in \mathcal{T}_\mathcal{I}}$ and $(W_s)_{s \in \mathcal{T}_\mathcal{J}}$ are called row and column cluster bases.

Remark 11 (Storage) An \mathcal{H}^2 -matrix is represented by its nested cluster bases, its $k \times k$ coupling matrices $(S_b)_{b \in \mathcal{L}_{\mathcal{I} \times \mathcal{J}}^+}$ and its nearfield matrices $(G_{|\hat{t} \times \hat{s}})_{b \in \mathcal{L}_{\mathcal{I} \times \mathcal{J}}^-}$. We have already seen in Remark 9 that the nested representations of the cluster bases require $\mathcal{O}(n_\mathcal{I} k)$ and $\mathcal{O}(n_\mathcal{J} k)$ units of storage, respectively. The coupling matrices require $\mathcal{O}(k^2)$ units of storage per block, leading to total requirements of $\mathcal{O}(n_\mathcal{I} k)$ for a sparse block tree $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$. For $(t, s) = b \in \mathcal{L}_{\mathcal{I} \times \mathcal{J}}^-$ both t and s are leaf clusters and so $\# \hat{t}$ and $\# \hat{s}$ are small, usually bounded by k , and we can conclude that the nearfield matrices require $\mathcal{O}(n_\mathcal{I} k)$ units of storage if $\mathcal{T}_{\mathcal{I} \times \mathcal{J}}$ is sparse. In total an \mathcal{H}^2 -matrix representation requires only $\mathcal{O}((n_\mathcal{I} + n_\mathcal{J}) k)$ units of storage [9, 8].

Approximating an arbitrary matrix $X \in \mathbb{R}^{\mathcal{I} \times \mathcal{J}}$ by an \mathcal{H}^2 -matrix becomes a relatively simple task if we apply orthogonal projections. These projections are readily available if the cluster bases are *orthogonal*:

Definition 12 (Orthogonal cluster basis) We call a cluster basis $(V_t)_{t \in \mathcal{T}_\mathcal{G}}$ orthogonal if

$$V_t^T V_t = I \quad \text{for all } t \in \mathcal{T}_\mathcal{G}.$$

If $(V_t)_{t \in \mathcal{T}_\mathcal{G}}$ and $(W_s)_{s \in \mathcal{T}_\mathcal{G}}$ are orthogonal cluster bases, the optimal coupling matrices for a given matrix G (with respect both to the Frobenius norm and the spectral norm) are given by

$$S_b := V_t^T G|_{\hat{t} \times \hat{s}} W_s \quad \text{for all } b = (t, s) \in \mathcal{L}_{\mathcal{T} \times \mathcal{S}}^+. \quad (4)$$

This property can be used to compute the best approximation of the product of \mathcal{H}^2 -matrices in $\mathcal{O}(nk^2)$ operations [5] as long as both cluster bases are known in advance. Unfortunately the suitable cluster bases for the results of arithmetic operations are typically not known. Thus we have to construct adaptive cluster bases during the computations, see section 2.5 and [6, 8, 9].

2.2 Algebraic operations

We want to compute the eigenvalues of a matrix $A \in \mathbb{R}^{\mathcal{I} \times \mathcal{I}}$ corresponding to a Galerkin discretization of an elliptic partial differential equation via a slicing method. This method relies on a sufficiently accurate approximation of the LDL^T factorization of shifted matrices.

In order to construct an approximation of this factorization, we employ an algorithm based on low-rank updates [10]. We assume for the sake of simplicity that every non-leaf cluster has exactly two sons. We obtain the following block equation for the LDL^T factorization of a submatrix $A|_{\hat{t} \times \hat{t}}$ for non-leaf clusters t with $\text{sons}(t) = \{t_1, t_2\}$:

$$\begin{aligned} \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} &= A_{\hat{t} \times \hat{t}} = L_{\hat{t} \times \hat{t}} D_{\hat{t} \times \hat{t}} L_{\hat{t} \times \hat{t}}^T \\ &= \begin{pmatrix} L_{11} & \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} D_{11} & \\ & D_{22} \end{pmatrix} \begin{pmatrix} L_{11} & L_{21}^T \\ & L_{22} \end{pmatrix} \\ &= \begin{pmatrix} L_{11} D_{11} L_{11}^T & L_{11} D_{11} L_{21}^T \\ L_{21} D_{11} L_{11}^T & L_{21} D_{11} L_{21}^T + L_{22} D_{22} L_{22}^T \end{pmatrix}. \end{aligned}$$

We can solve $A_{11} = L_{11} D_{11} L_{11}^T$ by recursion to get L_{11} and D_{11} . If the recursion reaches a leaf block, the block is a sufficiently small matrix in standard representation and the LDL^T factorization can be computed by standard algorithms.

In the second step we can obtain L_{21} by solving the triangular system $A_{12} = L_{11} D_{11} L_{21}^T$. This requires forward substitution for $A_{12} = L_{11} Y$ and solving the diagonal system $Y = D_{11} L_{21}^T$. The same block equation approach as above reduces the forward substitution to matrix-matrix multiplications of the form $C \leftarrow C + \alpha AB$.

Finally we can solve $A_{22} - L_{21} D_{11} L_{21}^T = L_{22} D_{22} L_{22}^T$ to get L_{22} and D_{22} . This means a matrix-matrix multiplication of the form $C \leftarrow C + \alpha AB$ and a recursion as in step one.

The block equation approach for the matrix-matrix multiplication $C \leftarrow C + \alpha AB$ leads to recursive calls $C_{ij} \leftarrow C_{ij} + \alpha A_{ik} B_{kj}$. The basis case of the recursion is when A or B is a leaf. Admissible leaves have low rank because of their three-term representation. Inadmissible leaves have low rank because they are small. In both cases we can compute a low rank representation XY^T of the product AB in linear complexity.

Altogether the arithmetic is reduced to the task of applying low-rank updates $C|_{\hat{t} \times \hat{t}} + XY^T$ to a submatrix of an \mathcal{H}^2 -matrix, where $X \in \mathbb{R}^{\hat{t} \times k}$ and $Y \in \mathbb{R}^{\hat{s} \times k}$.

2.3 \mathcal{H}^2 -matrix Representation of $C + XY^T$

In order to handle low-rank updates to \mathcal{H}^2 -matrices efficiently, we follow the approach described in [10], i.e., we consider $C + XY^T$ as an \mathcal{H}^2 -matrix with increased rank and apply the recompression algorithm [9] in order to reduce the rank while guaranteeing a given accuracy. We only outline the algorithm here for the sake of completeness and refer readers to [10] for details.

We first consider a *global* low-rank update $C \leftarrow C + XY^T$ and start by examining the \mathcal{H}^2 -matrix representation of the new matrix $\tilde{C} := C + XY^T$. For each admissible leaf $b = (t, s) \in \mathcal{L}_{\mathcal{I} \times \mathcal{J}}^+$, we obtain the following simple three-term representation:

$$\begin{aligned}\tilde{C}_{|\hat{t} \times \hat{s}} &= (C + XY^T)_{|\hat{t} \times \hat{s}} = V_t S_b W_s^T + X_{|\hat{t} \times k} Y_{|\hat{s} \times k}^T \\ &= \begin{pmatrix} V_t & X_{|\hat{t} \times k} \end{pmatrix} \begin{pmatrix} S_b \\ I_k \end{pmatrix} \begin{pmatrix} W_s & Y_{|\hat{s} \times k} \end{pmatrix}^T.\end{aligned}$$

This leads to the new cluster bases

$$\tilde{V}_t = \begin{pmatrix} V_t & X_{|\hat{t} \times k} \end{pmatrix} \quad \text{and} \quad \tilde{W}_s = \begin{pmatrix} W_s & Y_{|\hat{s} \times k} \end{pmatrix}.$$

These are nested with transfer matrices

$$\tilde{E}_t = \begin{pmatrix} E_t & \\ & I_k \end{pmatrix} \quad \text{and} \quad \tilde{F}_s = \begin{pmatrix} F_s & \\ & I_k \end{pmatrix}.$$

The new nested cluster bases \tilde{V} and \tilde{W} together with coupling matrices

$$\tilde{S}_b = \begin{pmatrix} S_b & \\ & I_k \end{pmatrix}$$

for each $b \in \mathcal{L}_{\mathcal{I} \times \mathcal{J}}$ give us an *exact* \mathcal{H}^2 -matrix representation of $\tilde{C} = C + XY^T$.

The drawback of this representation is the doubled rank. We solve this problem by applying the recompression algorithm described in [6, 8]: we construct adaptive orthogonal cluster bases and then approximate the original matrix in the space defined by these bases (cf. (4)).

2.4 Weight Matrices

In order to keep the presentation simple we denote the \mathcal{H}^2 -matrix \tilde{C} in the following just by C , the corresponding row and column cluster bases by $(V_t)_{t \in \mathcal{T}_{\mathcal{I}}}$ and $(W_s)_{s \in \mathcal{T}_{\mathcal{J}}}$, their rank by k , the coupling matrices by $(S_b)_{b \in \mathcal{L}_{\mathcal{I} \times \mathcal{J}}^+}$, and the nearfield matrices by $(C_{|b})_{b \in \mathcal{L}_{\mathcal{I} \times \mathcal{J}}^-}$. In our algorithm, these matrices are constructed implicitly according to the equations given in the previous section.

The recompression algorithm is based on the method introduced in [9] using the refinements added in [6]: the original algorithm relies on approximations of certain submatrices of C , and since this is an \mathcal{H}^2 -matrix, these submatrices can be represented by compact *weight matrices*. Here we only briefly outline the concept and refer readers to [6] and [8, Chapter 6.6] for details.

We consider only the construction of a row basis, since a column basis can be obtained by applying the same algorithm to the transposed matrix C^T .

The cluster basis V_t is directly used for the representation of all admissible blocks $(t, s) \in \mathcal{L}_{\mathcal{T} \times \mathcal{J}}^+$. We collect the corresponding column clusters in the set

$$\text{row}(t) = \{s \in \mathcal{T}_{\mathcal{J}} \mid (t, s) \in \mathcal{L}_{\mathcal{T} \times \mathcal{J}}^+\}.$$

Because of the nested structure, V_t influences also blocks $(t^*, s) \in \mathcal{L}_{\mathcal{T} \times \mathcal{J}}^+$ connected to predecessors t^* of t . We denote the set of predecessors by

$$\text{pred}(t) := \begin{cases} \{t\} & \text{if } t = \text{root}(\mathcal{T}_{\mathcal{J}}), \\ \{t\} \cup \text{pred}(t^+) & \text{for } t^+ \in \mathcal{T}_{\mathcal{J}}, t \in \text{sons}(t^+). \end{cases}$$

For the construction of the new cluster basis $(Q_t)_{t \in \mathcal{T}_{\mathcal{J}}}$, we have to consider the set

$$\text{row}^*(t) = \bigcup_{t^* \in \text{pred}(t)} \text{row}(t^*).$$

Let $\text{row}(t) = \{s_1, \dots, s_{\sigma}\}$ and $\text{row}^*(t) = \{s_1, \dots, s_{\rho}\}$ with $\sigma = \#\text{row}(t)$ and $\rho = \#\text{row}^*(t)$. The part of C which is described (directly or indirectly) by V_t is

$$C_t := (C_{|\hat{t} \times \hat{s}_1} \quad \dots \quad C_{|\hat{t} \times \hat{s}_{\rho}}).$$

Using the approach of (4) we search for an orthogonal matrix Q_t with lower rank than V_t such that

$$Q_t Q_t^T C_t \approx C_t.$$

We want to reach this goal via singular value decomposition.

Computing the SVD of C_t directly would be too expensive, but we can introduce weight matrices to significantly reduce the number of operations: if for a matrix Z_t there is an orthogonal matrix P_t with

$$C_t = V_t Z_t^T P_t^T,$$

we call Z_t a weight matrix for C and t . Since V_t has only k columns, the same holds for Z_t , and using, e.g., a QR factorization leads to an upper triangular Z_t with k columns and not more than k rows. For the construction of the cluster basis, we are only interested in the left singular vectors and the singular values of C_t . Due to the orthogonality of P_t , these quantities can be obtained by computing only the SVD of $V_t Z_t^T$ instead of working with C_t . Since Z_t^T has not more than k columns, this approach leads to a significant reduction in the computational work.

We construct the weight matrices by a top-down recursion: for the root of $\mathcal{T}_{\mathcal{J}}$, the weight matrix can be computed directly. For a cluster $t \in \mathcal{T}_{\mathcal{J}} \setminus \{\text{root}(\mathcal{T}_{\mathcal{J}})\}$, we assume that a weight matrix Z_{t^+} for its father $t^+ \in \mathcal{T}_{\mathcal{J}}$ has already been computed and denote the corresponding orthogonal matrix by P_{t^+} . Since the cluster basis $(V_t)_{t \in \mathcal{T}_{\mathcal{J}}}$ is nested, we have

$$\begin{aligned} C_t &= (C_{|\hat{t} \times \hat{s}_1} \quad \dots \quad C_{|\hat{t} \times \hat{s}_{\sigma}} \quad C_{t^+|\hat{t} \times \mathcal{J}}) \\ &= (C_{|\hat{t} \times \hat{s}_1} \quad \dots \quad C_{|\hat{t} \times \hat{s}_{\sigma}} \quad (V_{t^+} Z_{t^+}^T P_{t^+}^T)_{|\hat{t} \times \mathcal{J}}) \\ &= (C_{|\hat{t} \times \hat{s}_1} \quad \dots \quad C_{|\hat{t} \times \hat{s}_{\sigma}} \quad V_{t^+|\hat{t} \times k} Z_{t^+}^T P_{t^+}^T) \end{aligned}$$

$$= (V_t S_{(t,s_1)} W_{s_1}^T \quad \dots \quad V_t S_{(t,s_\sigma)} W_{s_\sigma}^T \quad V_t E_t Z_{t^+}^T P_{t^+}^T) = V_t B_t \quad (5)$$

with the matrix

$$B_t := (S_{(t,s_1)} W_{s_1}^T \quad \dots \quad S_{(t,s_\sigma)} W_{s_\sigma}^T \quad E_t Z_{t^+}^T P_{t^+}^T).$$

This allows us to obtain the following factorized representation of C_t :

$$\begin{aligned} C_t &= V_t B_t = V_t (S_{(t,s_1)} W_{s_1}^T \quad \dots \quad S_{(t,s_\sigma)} W_{s_\sigma}^T \quad E_t Z_{t^+}^T P_{t^+}^T) \\ &= V_t (S_{(t,s_1)} \quad \dots \quad S_{(t,s_\sigma)} \quad E_t Z_{t^+}^T) \begin{pmatrix} W_{s_1} & & & \\ & \ddots & & \\ & & W_{s_\sigma} & \\ & & & P_{t^+} \end{pmatrix}^T \\ &= V_t \tilde{Z}_t^T \tilde{P}_t^T. \end{aligned} \quad (6)$$

We assume in the following that the cluster basis W is orthogonal. If it is not, we can apply recursive QR factorizations to replace it by an orthogonal basis in linear complexity [11, Section 3.2]. Then \tilde{P}_t is orthogonal and \tilde{Z}_t is a weight matrix, but the number of rows of \tilde{Z}_t typically exceeds the number of columns. Thus we compute a thin QR decomposition $\tilde{Z}_t = \hat{P}_t \hat{Z}_t$ and get

$$C_t = V_t \tilde{Z}_t^T \tilde{P}_t^T = V_t \hat{Z}_t^T \hat{P}_t^T = V_t Z_t^T P_t^T.$$

P_t is orthogonal, and so Z_t is a small $k \times k$ weight matrix.

Altogether we can compute the weight matrices by a top down algorithm which only assembles \tilde{Z}_t and computes its QR decomposition. Only $k \times k$ weight matrices Z_t are stored and the number of considered blocks σ is bounded by the constant c_{sp} . Thus the storage requirement for one cluster $t \in \mathcal{T}_{\mathcal{J}}$ is in $\mathcal{O}(k^2)$ and the computational time is in $\mathcal{O}(k^3)$. The storage requirement for all weight matrices is in $\mathcal{O}(k^2 \# \mathcal{T}_{\mathcal{J}})$ and the computational time for the whole algorithm is in $\mathcal{O}(k^3 \# \mathcal{T}_{\mathcal{J}})$ [6, 8]. Using the standard assumption $\# \mathcal{T}_{\mathcal{J}} \lesssim n_{\mathcal{J}}/k$, we conclude that $\mathcal{O}(n_{\mathcal{J}} k)$ units of storage and $\mathcal{O}(n_{\mathcal{J}} k^2)$ operations are sufficient to set up all weight matrices.

2.5 Adaptive Cluster Basis

The weight matrices can be computed efficiently by a top-down traversal of the cluster tree $\mathcal{T}_{\mathcal{J}}$. Once they are at our disposal, we can use a bottom-up traversal of the cluster tree to construct the required adaptive cluster basis $(Q_t)_{t \in \mathcal{T}_{\mathcal{J}}}$ following the method given in [6] and [8, Chapter 6.6].

With the help of the weight matrices we get

$$\begin{aligned} \|Q_t Q_t^T C_t - C_t\| &= \|Q_t Q_t^T V_t Z_t^T P_t^T - V_t Z_t^T P_t^T\| \\ &= \|Q_t Q_t^T V_t Z_t^T - V_t Z_t^T\| \end{aligned} \quad (7)$$

for both the spectral and the Frobenius norm. Thus we only have to compute the SVD of $V_t Z_t^T$ instead of C_t . The direct approach would have linear complexity in each cluster and we would end up with log-linear complexity due to Remark 3. We also would not obtain a nested cluster basis.

In order to avoid both problems, we take advantage of the nested structure of $(V_t)_{t \in \mathcal{T}_\mathcal{G}}$ and $(Q_t)_{t \in \mathcal{T}_\mathcal{G}}$. We arrange the computation of the adaptive cluster basis $(Q_t)_{t \in \mathcal{T}_\mathcal{G}}$ in a bottom-up algorithm that also computes the basis change matrices $R_t := Q_t^T V_t$ for all $t \in \mathcal{T}_\mathcal{G}$ that can be used to compute the new coupling matrices efficiently.

In leaf clusters we compute the SVD of $V_t Z_t^T$ directly and use the left singular vectors corresponding to the k largest singular values to construct the orthogonal matrix Q_t . The computational time for each leaf is $O(k^2 \#t)$ and for all leaves together it is in $O(n_\mathcal{G} k^2)$ (see Remark 2).

The cluster basis in a non-leaf cluster is given by the nested representation

$$V_t = \begin{pmatrix} V_{t_1} E_{t_1} \\ V_{t_2} E_{t_2} \end{pmatrix}.$$

We assume that the matrices Q_{t_1} and Q_{t_2} for the sons have already been computed, and the nested structure of $(Q_t)_{t \in \mathcal{T}_\mathcal{G}}$ implies that anything that cannot be represented by these matrices also cannot be represented by Q_t , so applying a projection to the range of the son matrices does not change the quality of the approximation. If we let

$$U_t := \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix},$$

the orthogonal projection is given by $U_t U_t^T$ and applying it to V_t yields

$$U_t U_t^T V_t = U_t \begin{pmatrix} Q_{t_1}^T & \\ & Q_{t_2}^T \end{pmatrix} \begin{pmatrix} V_{t_1} E_{t_1} \\ V_{t_2} E_{t_2} \end{pmatrix} = U_t \begin{pmatrix} R_{t_1} E_{t_1} \\ R_{t_2} E_{t_2} \end{pmatrix} = U_t \widehat{V}_t$$

with a $(2k) \times k$ matrix $\widehat{V}_t = U_t^T V_t$. We compute the SVD of $\widehat{V}_t Z_t^T$ and again use the left singular vectors corresponding to the k largest singular values to form an orthogonal matrix $\widehat{Q}_t \in \mathbb{R}^{(2k) \times k}$. The new cluster basis is defined by $Q_t := U_t \widehat{Q}_t$. We deduce with Pythagoras' identity

$$\begin{aligned} & \|Q_t Q_t^T V_t Z_t^T - V_t Z_t^T\|^2 \\ &= \|U_t \widehat{Q}_t \widehat{Q}_t^T U_t^T V_t Z_t^T - U_t U_t^T V_t Z_t^T\|^2 \\ &+ \|U_t U_t^T V_t Z_t^T - V_t Z_t^T\|^2 \\ &= \|\widehat{Q}_t \widehat{Q}_t^T \widehat{V}_t Z_t^T - \widehat{V}_t Z_t^T\|^2 + \|U_t U_t^T V_t Z_t^T - V_t Z_t^T\|^2. \end{aligned} \tag{8}$$

Thus the error for the cluster t can be bounded by the error of the projection of the son clusters and the error of the truncated SVD of $\widehat{V}_t Z_t^T$. We will investigate the error in subsection 2.6.

The basis change matrix R_t is computed in $\mathcal{O}(k^3)$ operations via

$$R_t = Q_t^T V_t = \widehat{Q}_t^T U_t^T V_t = \widehat{Q}_t^T \widehat{V}_t.$$

The transfer matrices of Q_t can be constructed by splitting \widehat{Q}_t into its lower and upper half, i.e., by using

$$Q_t = U_t \widehat{Q}_t = \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} \begin{pmatrix} F_{t_1} \\ F_{t_2} \end{pmatrix}.$$

We can see that leaf clusters $t \in \mathcal{L}_\mathcal{G}$ require $\mathcal{O}(k^2 \#t)$ operations while non-leaf clusters $t \in \mathcal{T}_\mathcal{G} \setminus \mathcal{L}_\mathcal{G}$ require $\mathcal{O}(k^3)$. The total computational time of the algorithm therefore is in $\mathcal{O}(k^2 n_\mathcal{G} + k^3 \# \mathcal{T}_\mathcal{G})$ due to Remark 2. By the standard assumption $\# \mathcal{T}_\mathcal{G} \lesssim n_\mathcal{G} / k$, we conclude that not more than $\mathcal{O}(n_\mathcal{G} k^2)$ operations are required to construct the new cluster basis [6, 8].

2.6 Error Control

As we have seen in the previous subsections we are able to recompress an \mathcal{H}^2 -matrix in linear complexity and (8) suggests that the resulting error can be controlled by the accuracy of the truncated SVD. In this section, we describe a simplified version of the blockwise error control strategy developed in [6] that, according to our experiments, is suitable for treating eigenvalue problems.

Let $b = (t, s) \in \mathcal{L}_{\mathcal{J} \times \mathcal{J}}^+$. Multiplying the matrices in (8) by P_t^T from the right, using $C_t = V_t B_t = V_t Z_t^T P_t^T$, and restricting to $\hat{t} \times \hat{s}$, we obtain

$$\begin{aligned} & \|Q_t Q_t^T C_{|\hat{t} \times \hat{s}} - C_{|\hat{t} \times \hat{s}}\|^2 \\ &= \|Q_t Q_t^T V_t B_{t|k \times \hat{s}} - V_t B_{t|k \times \hat{s}}\|^2 \\ &= \|\widehat{Q}_t \widehat{Q}_t^T \widehat{V}_t B_{t|k \times \hat{s}} - \widehat{V}_t B_{t|k \times \hat{s}}\|^2 \\ &\quad + \|U_t U_t^T V_t B_{t|k \times \hat{s}} - V_t B_{t|k \times \hat{s}}\|^2. \end{aligned} \tag{9}$$

Due to the nested structure of V_t and the definition of U_t , we have

$$\begin{aligned} & \|U_t U_t^T V_t B_{t|k \times \hat{s}} - V_t B_{t|k \times \hat{s}}\|^2 \\ &= \|Q_{t_1} Q_{t_1}^T V_{t_1} E_{t_1} B_{t|k \times \hat{s}} - V_{t_1} E_{t_1} B_{t|k \times \hat{s}}\|^2 \\ &\quad + \|Q_{t_2} Q_{t_2}^T V_{t_2} E_{t_2} B_{t|k \times \hat{s}} - V_{t_2} E_{t_2} B_{t|k \times \hat{s}}\|^2 \\ &= \|Q_{t_1} Q_{t_1}^T V_{t_1} B_{t_1|k \times \hat{s}} - V_{t_1} B_{t_1|k \times \hat{s}}\|^2 \\ &\quad + \|Q_{t_2} Q_{t_2}^T V_{t_2} B_{t_2|k \times \hat{s}} - V_{t_2} B_{t_2|k \times \hat{s}}\|^2. \end{aligned}$$

By simple induction we get

$$\begin{aligned} & \|Q_t Q_t^T C_{|\hat{t} \times \hat{s}} - C_{|\hat{t} \times \hat{s}}\|^2 \\ &= \sum_{r \in \text{sons}^*(t)} \|\widehat{Q}_r \widehat{Q}_r^T \widehat{V}_r B_{r|k \times \hat{s}} - \widehat{V}_r B_{r|k \times \hat{s}}\|^2 \end{aligned} \tag{10}$$

with the set of descendants given by

$$\text{sons}^*(t) := \begin{cases} \{t\} & \text{if } t \in \mathcal{L}_{\mathcal{J}} \\ \{t\} \cup \bigcup_{t' \in \text{sons}(t)} \text{sons}^*(t') & \text{otherwise} \end{cases}$$

and extending the notation to $\widehat{Q}_t = Q_t$ and $\widehat{V}_t = V_t$ for leaf clusters $t \in \mathcal{L}_{\mathcal{J}}$.

Equation (10) provides us with an explicit error representation. We get an efficiently computable error bound by extending $B_{r|k \times \hat{s}}$ to the larger matrix B_r and using $B_r = Z_r^T P_r^T$ to reduce to the weight matrix:

$$\begin{aligned} & \|Q_t Q_t^T C_{|\hat{t} \times \hat{s}} - C_{|\hat{t} \times \hat{s}}\|^2 \\ &= \sum_{r \in \text{sons}^*(t)} \|\widehat{Q}_r \widehat{Q}_r^T \widehat{V}_r B_{r|k \times \hat{s}} - \widehat{V}_r B_{r|k \times \hat{s}}\|^2 \\ &\leq \sum_{r \in \text{sons}^*(t)} \|\widehat{Q}_r \widehat{Q}_r^T \widehat{V}_r B_r - \widehat{V}_r B_r\|^2 \end{aligned}$$

$$= \sum_{r \in \text{sons}^*(t)} \|\widehat{Q}_r \widehat{Q}_r^T \widehat{V}_r Z_r^T - \widehat{V}_r Z_r^T\|^2.$$

This is an error bound that we can control directly via the truncation criterion of the SVD used to compute \widehat{Q}_r . Unfortunately it does not give us direct error control for individual blocks, which is crucial for efficient and reliable algebraic operations. If we could bound each term in (10) by

$$\begin{aligned} & \|\widehat{Q}_r \widehat{Q}_r^T \widehat{V}_r B_{r|k \times \hat{s}} - \widehat{V}_r B_{r|k \times \hat{s}}\|^2 \\ & \leq \frac{\varepsilon^2}{3} \|C_{|\hat{t} \times \hat{s}}\|^2 \left(\frac{1}{3}\right)^{\text{level}(r) - \text{level}(t)}, \end{aligned}$$

we would get

$$\begin{aligned} & \|Q_t Q_t^T C_{|\hat{t} \times \hat{s}} - C_{|\hat{t} \times \hat{s}}\|^2 \\ & \leq \frac{\varepsilon^2}{3} \|C_{|\hat{t} \times \hat{s}}\|^2 \sum_{r \in \text{sons}^*(t)} \left(\frac{1}{3}\right)^{\text{level}(r) - \text{level}(t)} \\ & = \frac{\varepsilon^2}{3} \|C_{|\hat{t} \times \hat{s}}\|^2 \sum_{\ell = \text{level}(t)}^{p_{\mathcal{J}}} \left(\frac{1}{3}\right)^{\ell - \text{level}(t)} \#\{r \in \text{sons}^*(t) : \text{level}(r) = \ell\} \\ & \leq \frac{\varepsilon^2}{3} \|C_{|\hat{t} \times \hat{s}}\|^2 \sum_{\ell = \text{level}(t)}^{p_{\mathcal{J}}} \left(\frac{2}{3}\right)^{\ell - \text{level}(t)} < \varepsilon^2 \|C_{|\hat{t} \times \hat{s}}\|^2 \end{aligned} \quad (11)$$

by the geometric summation formula.

We cannot simply set the tolerance in each cluster $r \in \text{sons}^*(t)$ to

$$\omega_{r,b}^2 := \frac{\varepsilon^2}{3} \|C_{|\hat{t} \times \hat{s}}\|^2 \left(\frac{1}{3}\right)^{\text{level}(r) - \text{level}(t)}$$

because it depends not only on r , but also on b . The solution is to put the factor $\omega_{r,b}$ into the weight matrix [6]. The condition

$$\|\widehat{Q}_r \widehat{Q}_r^T \widehat{V}_r B_{r|k \times \hat{s}} - \widehat{V}_r B_{r|k \times \hat{s}}\|^2 \leq \omega_{r,b}^2$$

is equivalent to

$$\|\widehat{Q}_r \widehat{Q}_r^T \widehat{V}_r \omega_{r,b}^{-1} B_{r|k \times \hat{s}} - \widehat{V}_r \omega_{r,b}^{-1} B_{r|k \times \hat{s}}\|^2 \leq 1.$$

Since $\omega_{r',b} = \omega_{r,b}/3$ holds for all $r \in \text{sons}^*(t)$ and $r' \in \text{sons}(r)$, we can include the factors in the algorithm for constructing the weight matrices in (6) and get

$$\widetilde{Z}_{t,\omega}^T = \begin{pmatrix} \omega_{r,(t,s_1)}^{-1} S_{(t,s_1)} & \cdots & \omega_{r,(t,s_\sigma)}^{-1} S_{(t,s_\sigma)} & 3E_t Z_{t^+}^T \end{pmatrix}. \quad (12)$$

The resulting weight matrices $Z_{t,\omega}$ satisfy

$$(Z_{t,\omega}^T P_t^T)_{|k \times \hat{s}} = \omega_{r,b}^{-1} B_{r|k \times \hat{s}},$$

therefore we get the error bound in (11) if we replace Z_t by $Z_{t,\omega}$ and ensure that the rank k used in the truncation is large enough to capture all singular values larger than one.

Now we have found a recompression algorithm with linear complexity $\mathcal{O}(n_{\mathcal{J}} k^2)$ allowing us to control the relative error in each admissible block both in the spectral and the Frobenius norm. The next subsection shows that we can generalize our approach to local updates without losing the optimal complexity.

2.7 Algorithmic Challenges of Local Updates

Local updates $C_{\hat{t}_0 \times \hat{s}_0} \leftarrow C_{\hat{t}_0 \times \hat{s}_0} + XY^T$ of submatrices defined by a block $b_0 = (t_0, s_0) \in \mathcal{T}_{\mathcal{I}} \times \mathcal{J}$ pose a number of additional challenges in comparison with the global update discussed above. In order to obtain linear complexity with respect to the size of the local block, the top-down procedure of computing the weight matrices and the update of coupling matrices need to be investigated more closely. The first one requires the weight matrix of the father and so of all predecessors. The second task has to update all coupling matrices even if they are not in the sub-block of the update.

We go through four parts of the local update and discuss the special issues: the computation of the weight matrices, the construction of the adaptive cluster bases for $C + XY^T$, the update of the \mathcal{H}^2 -matrix, and the preparation of auxiliary data required for further updates.

The efficient computation of the weight matrix of a cluster requires the weight matrix of the father. If we compute an update for the root this poses no problem, but computing the weight for a higher-level cluster would require us to visit all of its predecessors and therefore lead to undesirable terms in the complexity estimate. We solve this problem by computing the weight matrices for all clusters in a preparation step. This can be done in linear complexity once before we start the LDL^T factorization. For the local update we only have to recompute the weight matrices in the sub-block of the update. Outside of the sub-block, the matrix remains unchanged, therefore we do not have to update the weight matrices.

There is a second challenge arising from the computation of the weight matrices. The blocks (t, s_i) corresponding to the matrices $C_{\hat{t}, \hat{s}_i}$ do not necessarily belong to the sub-block of the local update. Thus we need access to all admissible blocks (t, s_i) with row cluster t . This is handled by lists containing all row and column blocks connected to clusters.

As shown in subsection 2.5 the computation of the adaptive cluster basis is a bottom-up algorithm that can be applied to the subtree corresponding to the update. The cluster basis outside of this subtree remains unchanged. All predecessors can be updated by simply modifying the transfer matrix connecting the root of the subtree to its father. Hence there are no special problems for the local update in comparison to the global update.

The third step is more challenging than the second one. The coupling matrices have to be updated for all blocks (t, s_i) , i.e., they have to be multiplied by the basis change matrix R_t . Since s_i may lie outside of the subblock that is being updated, we again make use of the block lists mentioned before. In each of these blocks we only have to multiply the small matrices R_t and S_{t, s_i} . Assuming again that the block tree is c_{sp} -sparse, for one cluster $t \in \mathcal{T}_{\mathcal{I}}$ not more than c_{sp} such products have to be computed, so the number of operations is in $\mathcal{O}(k^3)$ for one cluster. Updating all blocks connected to the sons of t_0 or s_0 requires $\mathcal{O}(k^3(\#\mathcal{T}_{\hat{t}_0} + \#\mathcal{T}_{\hat{s}_0}))$ operations, where $\mathcal{T}_{\hat{t}_0}$ and $\mathcal{T}_{\hat{s}_0}$ denote the subtrees of $\mathcal{T}_{\mathcal{I}}$ and $\mathcal{T}_{\mathcal{J}}$ with roots t_0 and s_0 . Using again the standard assumptions $\#\mathcal{T}_{\hat{t}_0} \lesssim \#\hat{t}_0/k$ and $\#\mathcal{T}_{\hat{s}_0} \lesssim \#\hat{s}_0/k$, we obtain a complexity of $\mathcal{O}(k^2(\#\hat{t}_0 + \#\hat{s}_0))$.

To conclude the local update, we have to ensure that the weight matrices are correct by recomputing them in the subtree $\mathcal{T}_{\hat{t}_0}$ and $\mathcal{T}_{\hat{s}_0}$. The weight matrices do not change for clusters outside the sub-block.

Altogether we end up with computational complexity in $\mathcal{O}(k^2(\#\hat{t}_0 + \#\hat{s}_0))$ for the local update in a sub-block $b_0 = (t_0, s_0)$. Using this estimate, we can prove [10] that the matrix multiplication and other higher arithmetic functions require not more than $\mathcal{O}((n_{\mathcal{I}} + n_{\mathcal{J}})k^2 \log(n))$ operations.

3 Slicing the Spectrum

In order to use our efficient matrix-arithmetic operations to solve an eigenvalue problem, we use the slicing-the-spectrum algorithm that has been described in [24]. For the related \mathcal{H}_ℓ -matrices, which are \mathcal{H} -matrices with a particularly simple block tree, the algorithm has been investigated in [4]. Further, in [4] it has been shown by numerical examples that a generalization of the approach to \mathcal{H} -matrices does not lead to an efficient algorithm in general.

We are computing the eigenvalues of a symmetric matrix. Thus all eigenvalues are real and the function $v(\sigma) = \#\{\lambda \in \Lambda(A) : \lambda < \sigma\}$ is well defined for all $\sigma \in \mathbb{R}$. If $v(a) < m \leq v(b)$, we know that the interval $[a, b]$ contains the m -th smallest eigenvalue λ_m of A . We can run a bisection algorithm on this interval until the interval is small enough. The midpoint of the interval is then taken as approximation $\hat{\lambda}_m$ of the desired eigenvalue. We bisect the interval by computing $v(\frac{a+b}{2})$. If $v(\frac{a+b}{2}) > m$, we continue with $[a, \frac{a+b}{2}]$, otherwise with $[\frac{a+b}{2}, b]$. We stop the algorithm if $b - a < \varepsilon_{\text{ev}}$ holds. In our computations, we choose $\varepsilon_{\text{ev}} = 10^{-5}$.

It remains to explain how we get the inertia or $v(\sigma)$. The inertia is invariant under congruence transformations, thus the matrix D of the LDL^T factorization of A has the same inertia as A itself. To get $v(\sigma)$ we compute the LDL^T factorization of $A - \sigma I = L_\sigma D_\sigma L_\sigma^T$ and simply count the negative diagonal entries of D .

3.1 Accuracy

By using \mathcal{H}^2 -matrices, the computation of an LDL^T factorization is comparably cheap, taking essentially $\mathcal{O}(nk^2 \log n)$. This allows the fast computation of the inertia, which would be in $\mathcal{O}(n^3)$ for general dense matrices. The price we have to pay is that the factorization is only approximative, i.e., $A - \sigma I \approx \tilde{L}\tilde{D}\tilde{L}^T$, so we have to ensure that it is sufficiently accurate to yield the correct value $v(\sigma)$. In [24] it is shown that this is the case if $\|H_\sigma\| \leq \min_j |\lambda_j(A) - \sigma|$, with $H_\sigma := (A - \sigma I) - \tilde{L}_\sigma \tilde{D}_\sigma \tilde{L}_\sigma^T$.

Thus we need a bound for the error of the form $\|A - \tilde{L}\tilde{D}\tilde{L}^T\| \leq \delta\|A\|$. We further need this bound for all shifted matrices $A - \sigma I$. In the literature the LU-decomposition has received much more attention than the LDL^T factorization. Since the inertia of $A - \sigma I$ can also be obtained from an LU-decomposition, we will cite some results for LU-decompositions for hierarchical matrices: to our best knowledge such a bound is currently not available in the literature on \mathcal{H} - and \mathcal{H}^2 -matrices. In [2] it was shown that for certain \mathcal{H} -matrices originating from certain finite element discretizations there exist \mathcal{H} -matrices \tilde{L} and \tilde{U} so that $\|A - \tilde{L}\tilde{U}\| \leq \delta\|A\|$. This result has been generalized in [3, 22] and more recently in [13]. Unfortunately, it has so far not been shown that the algorithms actually used to compute approximations yield results satisfying similar estimates. Fortunately, many numerical experiments show that the algorithms for the computation of the \mathcal{H} -LU-decomposition are very good.

For the case of $A - \sigma I$, with $\sigma \neq 0$, the picture is not positive. In [4, Table 4.1] one can see that using shifts near eigenvalues leads to high local block ranks, which make the \mathcal{H} - LDL^T factorization expensive. We do not observe a similar behavior for \mathcal{H}^2 -arithmetic, but we cannot provide theoretical bounds for the ranks.

3.2 Generalized Eigenvalue Problem

For the solution of generalized eigenvalue problems we have to compute the inertia of $A - \sigma B$ instead of $A - \sigma I$. If we think of a finite element discretization as a basis for the generalized eigenvalue problem, then we observe that the structures of the mass and the stiffness matrix are similar enough to allow for a cheap computation of $A - \sigma B$ in the \mathcal{H}^2 -arithmetic. The mass matrix B can be stored as a sparse matrix. Fortunately, the nonzero entries in B correspond with inadmissible leaves in A , which are stored as dense matrices. Thus the subtraction $A - \sigma B$ affects only these inadmissible leaves.

Further research should investigate the numerical properties of the LDL^T factorization of $A - \sigma B$.

3.3 Parallelization

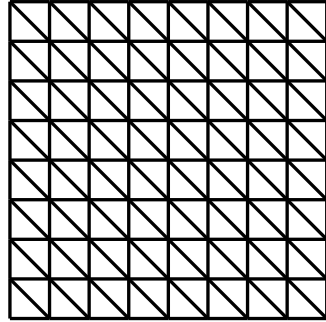
The slicing of disjoint intervals is independent, thus we can easily parallelize the algorithm by giving each node/core an instance of the matrix and an interval to slice. Since the size of the sparse matrix grows only linearly with the dimension of the matrix, this is possible for comparably large matrices. This simple parallelization has been used in [4] for the slicing algorithm for \mathcal{H}_ℓ -matrices. In [23] a speedup of 267 by using 384 processes has been reported for a MPI-based parallelization of the algorithm from [4]. For this parallelization a master-slave structure is used. The master provides each slave with a small interval, which the slaves slice until all eigenvalues are found. For these intervals the master provides a lower bound and an upper bound and the number of eigenvalues to be found. To provide this information some initial computations of $v(\sigma)$ are necessary. These are also performed by the slaves. The time required for the slicing of one interval varies and thus the intervals are chosen small enough to allow for a load balancing.

This parallelization works best for many cores. If the number of processes is small, the master process is frequently just waiting for answers, thus running 5 process on the quad-core CPU is improving the overall run-time.

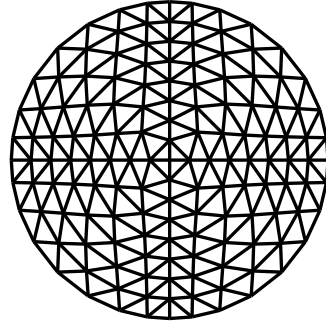
4 Numerical Experiments

Due to the facts described in Subsection 3.1 we cannot prove that the proposed algorithm is accurate and efficient; at least at the moment. Thus, numerical experiments are the only way to provide evidence that the slicing algorithm is performing well. For the numerical experiment we use the software package H2Lib developed by the Scientific Computing Group at Kiel University. This library provides examples of finite element discretizations on different triangle meshes, see Figure 1. These meshes can be refined as needed. We use a hexa-core CPU, Intel Xeon E5645 (running at 2.40 GHz).

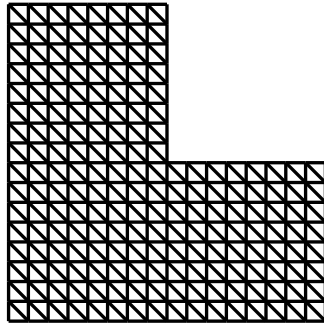
First, we want to show that the absolute accuracy of the computed eigenvalues is acceptable. Therefore we use the finite element matrix related to the meshed unit square. We refine the mesh in Figure 1(a) twice, compute the eigenvalues of this standard eigenvalue problem with the slicing algorithm and compare them to the actual eigenvalues, which are known exactly. In Figure 2 the accuracy of computed eigenvalues is shown. The computed eigenvalues lie all within the computed intervals.



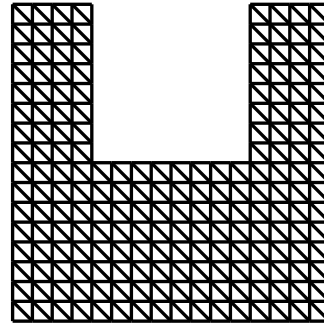
(a) Unit square.



(b) Unit circle.



(c) L-shape.



(d) U-shape.

Figure 1: Meshes for different geometries.

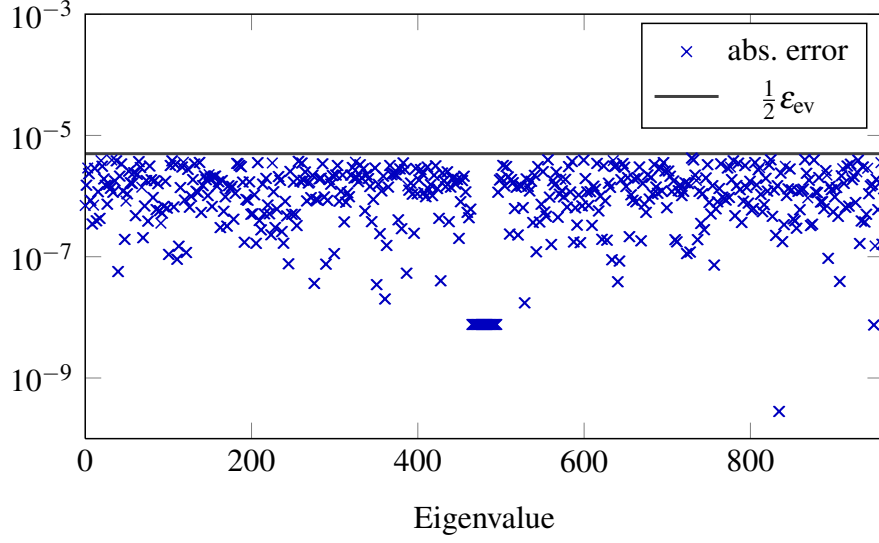


Figure 2: Absolute error $|\lambda_i - \hat{\lambda}_i|$ for a 961×961 finite element stiffness matrix corresponding to the unit square, $\varepsilon_{\text{ev}} = 10^{-5}$.

On the same mesh we then compute the mass matrix and solve the generalized eigenvalue problem, both with the LAPACK [1] eigenvalue solver for symmetric generalized eigenvalue problems `dsygv` and with the slicing algorithm. The result is similar to the previous one, as we observe in Figure 3 that again the allowed tolerance is fulfilled for all eigenvalues.

Since we are solving finite element eigenvalue problems, we expect the smallest eigenvalues to converge to the eigenvalues of the differential operator. This can be seen in Figure 4 for the 8 smallest eigenvalues, where 3 refinements correspond to the mesh shown in Figure 1(a): we obtain the $\mathcal{O}(h^2)$ convergence predicted by standard theory.

In Table 1 the runtime, the time for one slice, and the accuracy are shown for different refinements of the meshes in Figure 1. The accuracy is the maximum absolute error for the computed eigenvalues compared with the results from the LAPACK eigensolver `dsygv`. For matrices with $n \geq 5000$ the accuracy is not computed, since the dense matrices are too large and the computations with LAPACK would take too long. Figure 5 shows the time per degree of freedom using a logarithmic scale for n . It seems to suggest a complexity of $\mathcal{O}(n \log n)$ for large values of n , i.e., the effective rank k of the \mathcal{H}^2 -matrix approximation of the LDL^T factorization appears to be bounded independently of the mesh size.

In Table 2 we compare the algorithm with the slicing algorithm for \mathcal{H} -matrices described in [4]. Since the \mathcal{H} lib [20] is more optimized with respect to speed than the H2lib we choose to reimplement the algorithm from [4] in the H2lib for a fair comparison. Thereby we also generalized the algorithm to generalized eigenvalue problems. We see that the implementation based on \mathcal{H}^2 is slightly faster at the same accuracy.

However, using LAPACK `dsygv`, based on an implicit QZ algorithm on the dense matrix, would be much faster for the computation of all eigenvalues. A backward stable algorithm

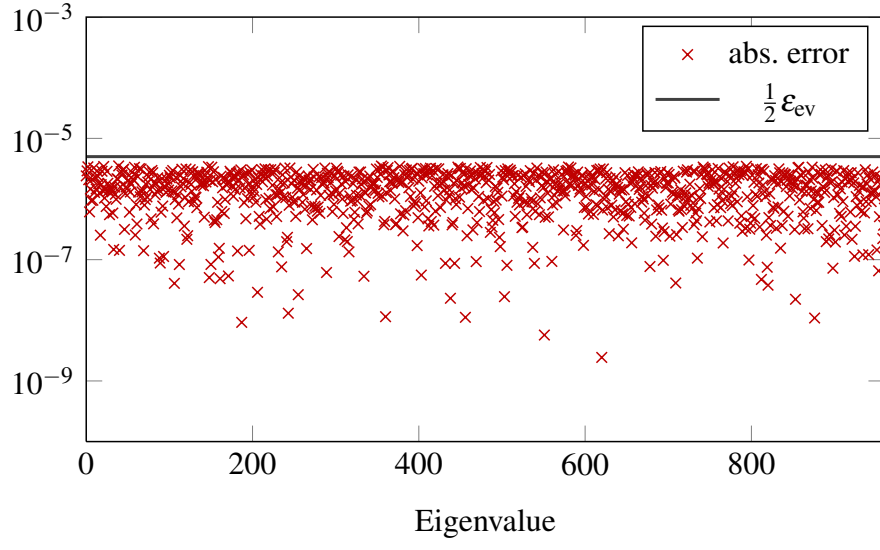


Figure 3: Absolute error $|\lambda_i - \tilde{\lambda}_i|$ for the generalized eigenproblem for a 961×961 finite element matrix corresponding to the unit square, $\varepsilon_{\text{ev}} = 10^{-5}$.

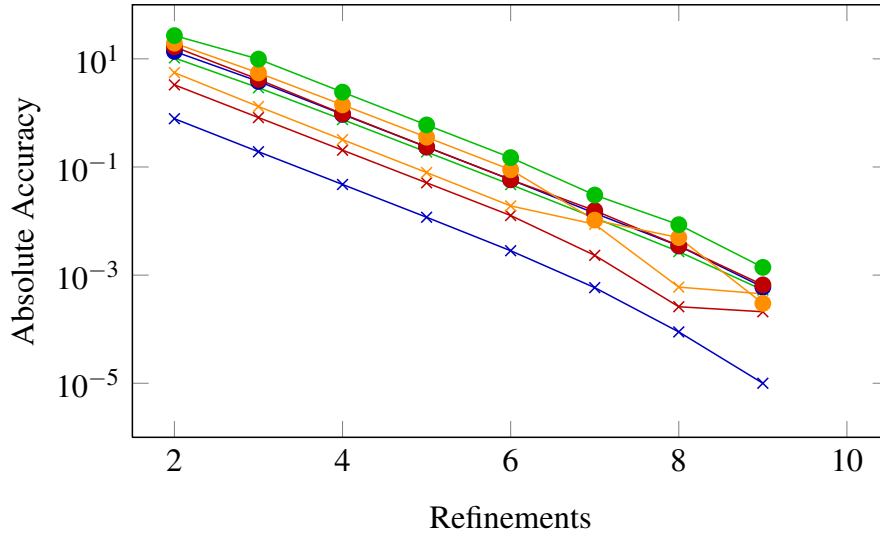


Figure 4: Convergence of the eigenvalues with respect to the mesh parameter.

Unit Square			
n	$t_{8 \text{ ev}}$ in s	$t_{\text{single slice}}$ in s	maximal abs. err.
225	0.23	< 0.01	3.1510_{-06}
961	2.18	0.01 ± 0.00	2.8435_{-06}
3,969	21.61	0.14 ± 0.01	3.2226_{-06}
16,129	190.20	1.30 ± 0.02	
65,025	1,304.22	9.31 ± 0.10	
261,121	7,577.24	56.08 ± 0.62	
1,046,529	39,992.06	305.03 ± 5.18	
Unit Circle			
n	$t_{8 \text{ ev}}$ in s	$t_{\text{single slice}}$ in s	maximal abs. err.
481	0.79	0.01 ± 0.01	2.3895_{-06}
1,985	9.02	0.06 ± 0.01	2.6512_{-06}
8,065	93.70	0.68 ± 0.02	
32,513	829.55	5.92 ± 0.11	
130,561	5,457.30	41.30 ± 0.59	
523,265	32,327.59	252.34 ± 2.75	
L-Shape			
n	$t_{8 \text{ ev}}$ in s	$t_{\text{single slice}}$ in s	maximal abs. err.
161	0.14	< 0.01	1.8366_{-06}
705	1.42	0.01 ± 0.00	3.8873_{-06}
2,945	12.33	0.07 ± 0.01	3.8643_{-06}
12,033	125.62	0.69 ± 0.01	
48,641	999.53	5.52 ± 0.06	
195,585	6,587.22	36.37 ± 0.37	
784,385	27,722.95	152.72 ± 5.84	
U-Shape			
n	$t_{8 \text{ ev}}$ in s	$t_{\text{single slice}}$ in s	maximal abs. err.
153	0.14	< 0.01	2.0762_{-06}
689	1.82	0.01 ± 0.00	3.7008_{-06}
2,913	10.14	0.06 ± 0.01	3.0300_{-06}
11,969	99.89	0.56 ± 0.01	
48,513	808.08	4.61 ± 0.04	
195,329	5,607.98	31.99 ± 0.36	
783,873	24,464.86	139.24 ± 5.04	

Table 1: Runtime for the computation of the 8 smallest eigenvalues on different shapes; for small matrices including the accuracy.

Unit Square, with mass matrix					
n	ev	$t_{\mathcal{H}}$	error	$t_{\mathcal{H}^2}$	error
225	8	0.24	3.15_{-06}	0.23	3.15_{-06}
961	8	2.60	2.84_{-06}	2.18	2.84_{-06}
3,969	8	29.07	3.22_{-06}	21.61	3.22_{-06}
16,129	8	246.83	—	190.20	—
65,025	8	1,555.23	—	1,304.22	—
261,121	8	—	—	7,577.24	—

Table 2: Comparing the \mathcal{H}^2 slicing algorithm with the \mathcal{H} slicing algorithm described in [4]. All timings in seconds.

Unit Square			
n	no. of cores	$t_{\text{all ev}}$ in s	speedup
961	1+0	185.40	single core code
961	1+1	182.73	1.01
961	2+1	99.85	1.86
961	3+1	68.71	2.70
961	4+1	47.77	3.88
961	5+1	38.60	4.80

Table 3: Speedup by parallelization; generalized eigenvalue problem, all eigenvalues.

is used to compute the eigenvalues to almost machine precision. The generalized eigenvalue problem, unit-square with mass matrix, of dimension 3969 can be solved in 38 s and the problem of dimension 16129 in 2296 s. The bigger problem requires about 2 GB storage. Thus one should only use the slicing algorithm for large problems.

Finally we test the MPI based parallelization, see Table 3. Here we use a quadcore CPU, Intel Core i5-3570 (running at 3.40 GHz) and compute the speedup in comparison with the runtime of the single core code. Since the master is not doing any work we see good speedups for up to 4 slave processes.

5 Conclusions

We have investigated whether the computation of eigenvalues of symmetric \mathcal{H}^2 -matrices can be done efficiently by slicing the spectrum. Our results show that for small n other methods, eventually even dense eigenvalue solver, are more efficient. However, the experiments further show that the computational costs per eigenvalue scale with $\mathcal{O}(n \log n)$ and thus for large n the method will be very efficient. It remains open whether the usage of \mathcal{H}^2 -arithmetic is significantly more efficient than \mathcal{H} -arithmetic or not. The additional structure might be used for higher efficiency, but produces also more overhead.

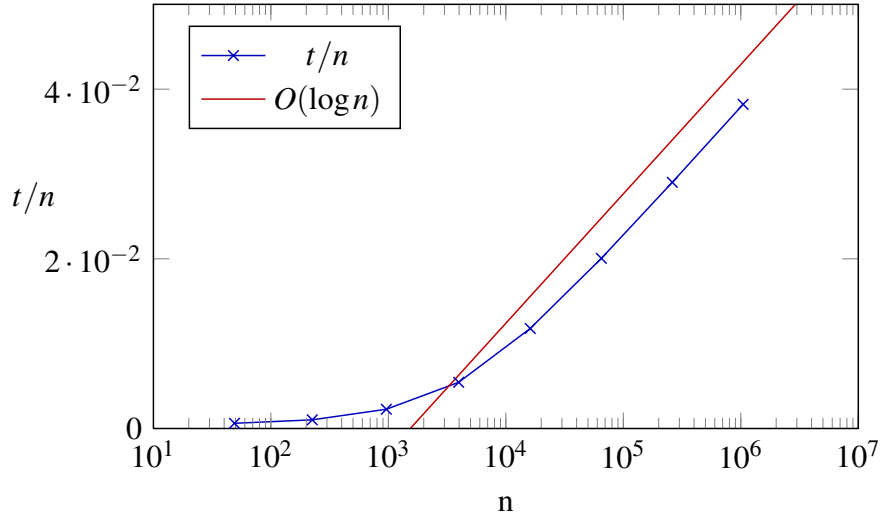


Figure 5: Runtime divided by matrix dimension; unit circle, 8 smallest eigenvalues of the generalized eigenvalue problem.

References

- [1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, third edition, 1999.
- [2] M. Bebendorf. Hierarchical LU decomposition-based preconditioners for BEM. *Computing*, 74(3):225–247, 2005.
- [3] M. Bebendorf. Why finite element discretizations can be factored by triangular hierarchical matrices. *SIAM J. Num. Anal.*, 45(4):1472–1494, 2007.
- [4] P. Benner and T. Mach. Computing all or some eigenvalues of symmetric \mathcal{H}_ℓ -matrices. *SIAM J. Sci. Comput.*, 34(1):A485–A496, 2012.
- [5] S. Börm. \mathcal{H}^2 -matrix arithmetics in linear complexity. *Computing*, 77(1):1–28, 2006.
- [6] S. Börm. Adaptive variable-rank approximation of dense matrices. *SIAM J. Sci. Comp.*, 30(1):148–168, 2008.
- [7] S. Börm. Approximation of solution operators of elliptic partial differential equations by \mathcal{H} - and \mathcal{H}^2 -matrices. *Numer. Math.*, 115(2):165–193, 2010.
- [8] S. Börm. *Efficient Numerical Methods for Non-local Operators: \mathcal{H}^2 -Matrix Compression, Algorithms and Analysis*, volume 14 of *EMS Tracts in Mathematics*. EMS, 2010.
- [9] S. Börm and W. Hackbusch. Data-sparse approximation by adaptive \mathcal{H}^2 -matrices. *Computing*, 69:1–35, 2002.

- [10] S. Börm and K. Reimer. Efficient arithmetic operations for rank-structured matrices based on hierarchical low-rank updates. Technical report, Institut für Informatik, Christian-Albrechts-Universität Kiel, feb 2014.
- [11] St. Börm. Construction of data-sparse \mathcal{H}^2 -matrices by hierarchical compression. *SIAM J. Sci. Comput.*, 31(3):1820–1839, 2009.
- [12] St. Börm, L. Grasedyck, and W. Hackbusch. Introduction to hierarchical matrices with applications. *Engineering Analysis with Boundary Elements*, 27:405–422, 2003.
- [13] M. Faustmann, J. Markus Melenk, and D. Praetorius. \mathcal{H} -matrix approximability of the inverses of FEM matrices. *ArXiv e-prints*, August 2013.
- [14] W. Hackbusch. A Sparse Matrix Arithmetic Based on \mathcal{H} -Matrices. Part I: Introduction to \mathcal{H} -Matrices. *Computing*, 62(2):89–108, 1999.
- [15] W. Hackbusch. *Hierarchische Matrizen. Algorithmen und Analysis*. Springer-Verlag, Berlin, 2009.
- [16] W. Hackbusch and M. Bebendorf. Existence of \mathcal{H} -Matrix Approximants to the Inverse FE-Matrix of elliptic Operators with L^∞ -Coefficients. *Numer. Math.*, 95:1–28, 2003.
- [17] W. Hackbusch, B. Khoromskij, and S. A. Sauter. On \mathcal{H}^2 -matrices. In *Lectures on Applied Mathematics: Proceedings of the Symposium Organized by the Sonderforschungsbereich 438 on the Occasion of Karl-Heinz Hoffmann's 60th Birthday, Munich, June 30-July 1, 1999*, page 9. Springer Verlag, 2000.
- [18] W. Hackbusch and B. N. Khoromskij. A sparse \mathcal{H} -matrix arithmetic. II. Application to multi-dimensional problems. *Computing*, 64(1):21–47, 2000.
- [19] W. Hackbusch, B. N. Khoromskij, and S. A. Sauter. On \mathcal{H}^2 -matrices. In H. Bungartz, R. Hoppe, and C. Zenger, editors, *Lectures on Applied Mathematics*, pages 9–29. Springer-Verlag, Berlin, 2000.
- [20] \mathcal{H} lib. <http://www.hlib.org>.
- [21] A. Knyazev, V. Mehrmann, and J. Xu, editors. *Numerical Solution of PDE Eigenvalue Problems*, volume 56. Mathematisches Forschungsinstitut Oberwolfach, 2013.
- [22] S. Le Borne, L. Grasedyck, and R. Kriemann. Domain-decomposition based \mathcal{H} -LU preconditioners. *Numer. Math.*, 112(4):565–600, 2009.
- [23] T. Mach. *Eigenvalue Algorithms for Symmetric Hierarchical Matrices*. Dissertation, Chemnitz University of Technology, 2012.
- [24] B.N. Parlett. *The Symmetric Eigenvalue Problem*. Prentice-Hall, Englewood Cliffs, first edition, 1980.